(12) UK Patent Application (19) GB (11) 2 366 426 (13) A

(43) Date of A Publication 06.03.2002

(21) Application No 0108828.5

(22) Date of Filing 09.04.2001

(30) Priority Data
(31) 09548109 (32) 12.04.2000 (33) US

(71) Applicant(s)
International Business Machines Corporation
(Incorporated in USA - New York)
Armonk, New York 10504, United States of America

(72) Inventor(s)
Gordon Taylor Davis
Marco C Heddes
Ross Boyd Leavens
Mark A Rinaldi

(74) Agent and/or Address for Service
IBM United Kingdom Limited
Intellectual Property Law, Hursley Park,
WINCHESTER, Hampshire, SO21 2JN,
United Kingdom

(51) INT CL^7
G06F 9/38 15/16 15/17

(52) UK CL (Edition T)
G4A AFGL

(56) Documents Cited
US 5507000 A US 4626634 A

(58) Field of Search
UK CL (Edition S) G4A AFGL AMP ANX
INT CL^7 G06F 9/38 15/16 15/17
Online: WPI, EPODOC, JAPIO

(54) Abstract Title
Multi-processor system with registers having a common address map

(57) A processor system comprises a core language processor 101, co-processors 107 - 111; each having special purpose, scalar 116 and array 117, registers; and an interface between the processors, where the interface maps the special purpose registers into a common address map. The system may be utilised as a protocol processor unit to provide instruction communication to a network, and the co-processors may compute CRC checksums, move data between local and main memories, search a tree structure, enqueue packets or assist in accessing the contents of registers. The interface may take the form of an execution interface 106 or a data interface 130.

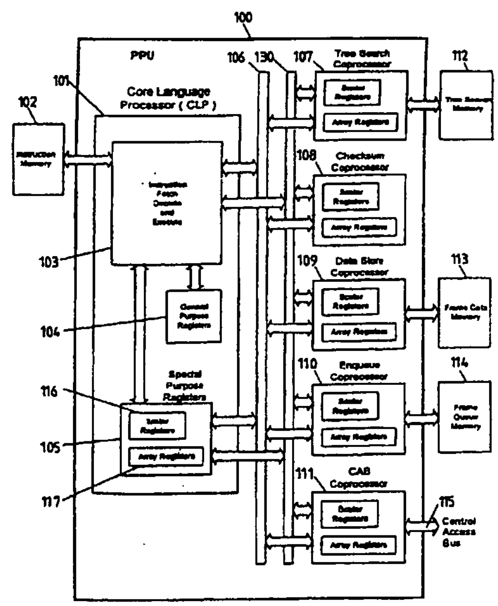
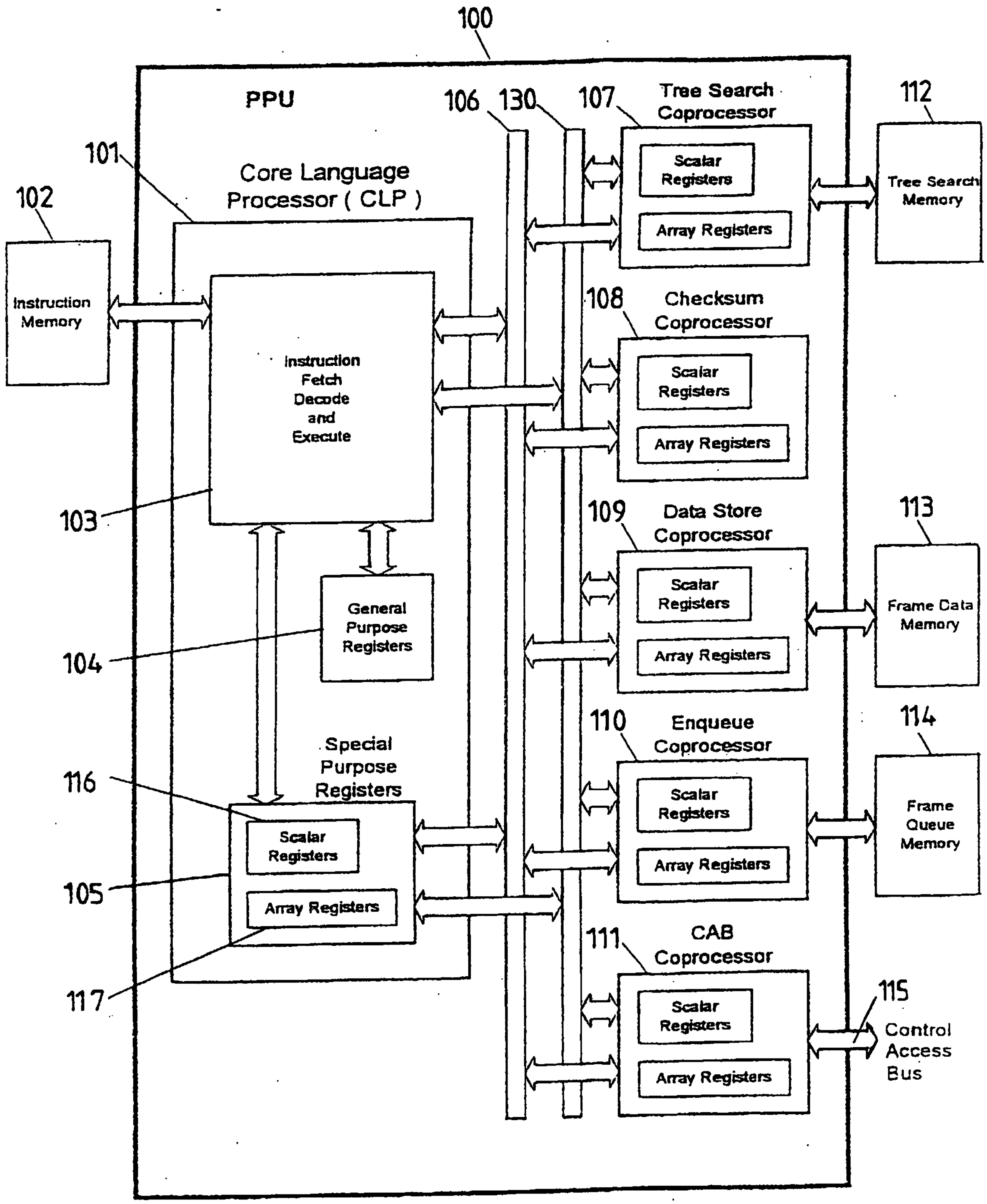


FIG. 1

GB 2 366 426 A



100
PPU
106
130
107
Tree Search Coprocessor
112
Scalar Registers
Array Registers
Tree Search Memory
101
Core Language Processor (CLP)
102
Instruction Memory
Instruction Fetch Decode and Execute
108
Checksum Coprocessor
Scalar Registers
Array Registers
103
109
Data Store Coprocessor
113
Scalar Registers
Array Registers
Frame Data Memory
104
General Purpose Registers
110
Enqueue Coprocessor
114
Special Purpose Registers
116
Scalar Registers
Scalar Registers
Array Registers
Frame Queue Memory
105
Array Registers
111
CAB Coprocessor
117
115
Scalar Registers
Control Access Bus
Array Registers

FIG. 1

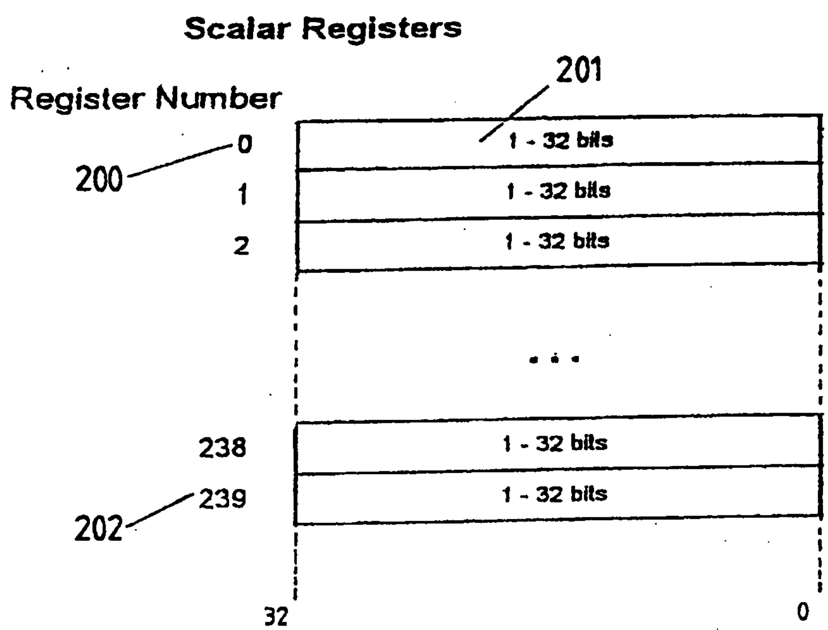

Scalar Registers
201
Register Number
0
200
1
2
1 - 32 bits
1 - 32 bits
1 - 32 bits
238
239
202
1 - 32 bits
1 - 32 bits
32
0

FIG. 2

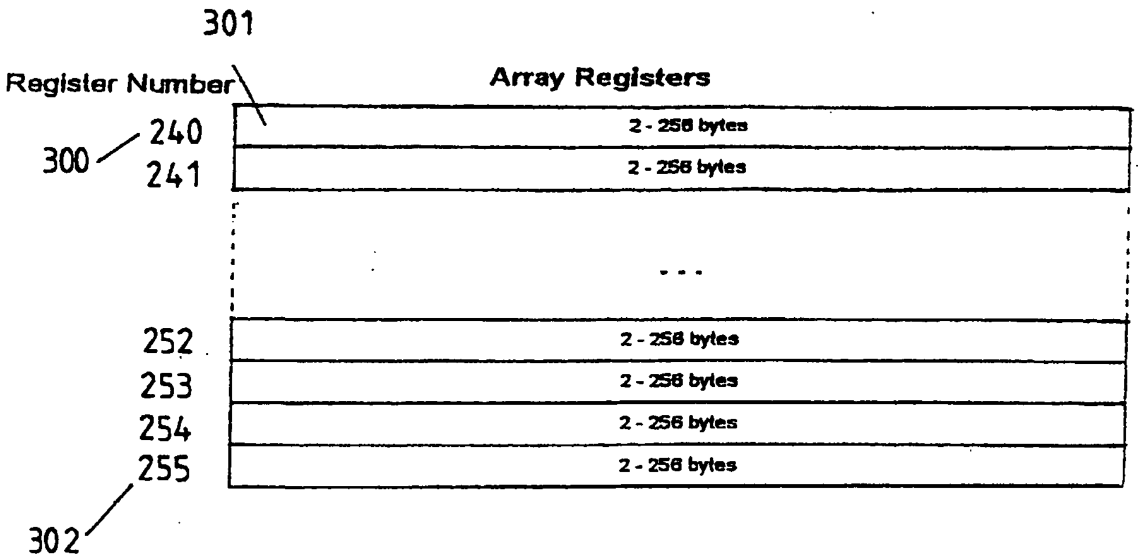

301
Register Number
Array Registers
300
240
241
252
253
254
255
302
2 - 256 bytes
2 - 256 bytes
...
2 - 256 bytes
2 - 256 bytes
2 - 256 bytes
2 - 256 bytes

FIG. 3a

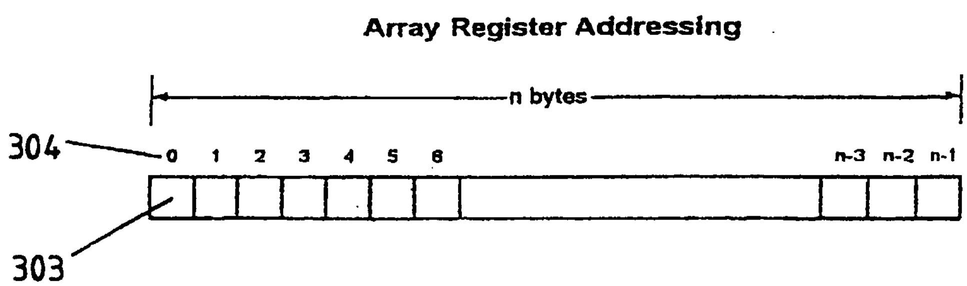

Array Register Addressing
n bytes
304
0 1 2 3 4 5 6
n-3 n-2 n-1
303

FIG. 3b

Ref.	Instruction	Encoding fields
	Invalid Operation	0 0 0 0 0 0 ; don't care
	Nop	0 0 0 0 0 1 ; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
	Exit	0 0 0 0 1 0 ; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
	Arithmetic Immediate	0 0 0 1 0 ; I ; OT ; Cond ; Rd ; imm4 ; AluOp ; imm8
	Arithmetic/Logical Register	0 0 0 1 1 ; I ; OT ; Cond ; Rd ; Rs ; AluOp ; 0 0 0 0 0 0 0 0
	Load Immediate	0 0 1 1 1 0 ; OT ; Cond ; Rd ; imm16
	Compare Immediate	0 0 1 1 1 1 ; OT ; Cond ; Rd ; imm16
400	Load / Store	0 0 1 0 0 ; x ; OT ; Cond ; R ; D ; Ra ; CP ; CR ; off6
401	Load / Store	0 0 1 0 1 ; x ; OT ; Cond ; R ; D ; 0 ; off2 ; CP ; CR ; off6
402	Load / Store	0 0 1 0 1 ; x ; OT ; Cond ; R ; D ; 1 ; 0 0 ; CP ; CR
	Branch and Link	0 0 1 1 0 0 ; 1 1 ; Cond ; Rt ; target16
	Return from subroutine	0 0 1 1 0 1 ; 0 0 ; Cond ; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
	Branch on Condition (register)	0 0 1 1 0 1 ; 0 1 ; Cond ; 0 0 0 0 ; Rt ; 0 0 0 0 0 0 0 0 0 0 0 0
	Branch on Condition (PC relative)	0 0 1 1 0 1 ; 1 0 ; Cond ; 0 0 0 0 ; disp16
	Branch on Condition (direct)	0 0 1 1 0 1 ; 1 1 ; Cond ; Rt ; target16
	Load Key	0 1 0 0 0 ; c ; bitoff5 ; len5 ; Rs ; 0 0 0 0 0 0 0 0 0 0 0 0
	Load Key	0 1 0 0 1 ; c ; 0 ; x ; bitoff 3 ; len5 ; 0 ; 0 ; off2 ; C#s ; CRs ; off6
	Load Key	0 1 0 0 1 ; c ; 0 ; x ; bitoff 3 ; len5 ; 1 ; Ra ; C#s ; CRs ; off6
	Load Key Immediate	0 1 0 0 1 ; c ; 1 ; 0 0 0 0 0 ; len4 ; imm16
	Logical Immediate	0 1 1 1 0 ; I ; Op ; Cond ; R ; imm16
	Move	1 0 0 ; W ; CP1 ; CR1 ; D ; Ra ; CP2 ; CR2 ; off6
	Move	1 0 1 ; W ; CP1 ; CR1 ; D ; 0 ; off2 ; CP2 ; CR2 ; off6
	Move	1 0 1 1 ; CP1 ; CR1 ; 0 ; 1 ; 0 0 ; CP2 ; CR2
	Move Multiple	1 1 0 0 ; CP1 ; CR1 ; 0 0 ; off4-1 ; 0 ; 0 0 0 ; CP2 ; CR2 ; 0 0 ; off4-2
	Move Immediate	1 1 0 1 ; CP1 ; CR1 ; imm16
408	Coprocessor Wait	1 1 1 0 ; 0 0 0 0 ; 0 0 0 0 0 0 0 0 ; 0 ; mask16
409	Coprocessor Wait and Branch	1 1 1 0 ; CP ; 0 0 0 0 0 0 ; ok ; 1 ; target16
410	Coprocessor Execute (direct)	1 1 1 1 ; CP ; Op ; a ; 0 ; imm16
411	Coprocessor Execute (indirect)	1 1 1 1 ; CP ; Op ; a ; 1 ; R ; imm12

FIG. 4

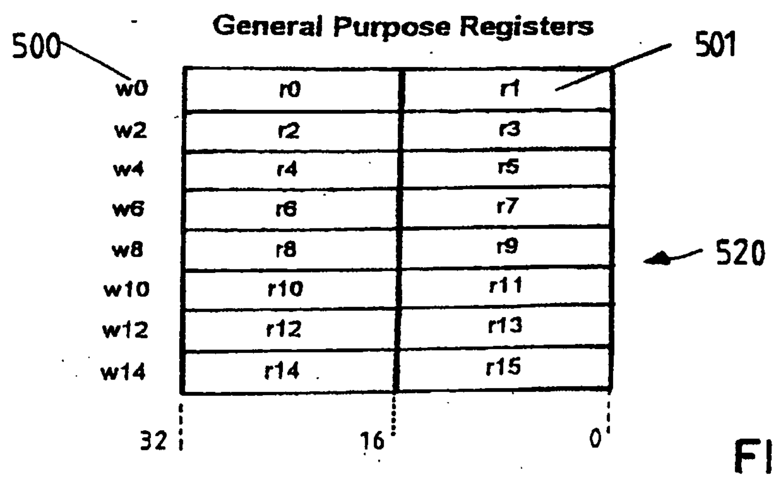


FIG. 5a

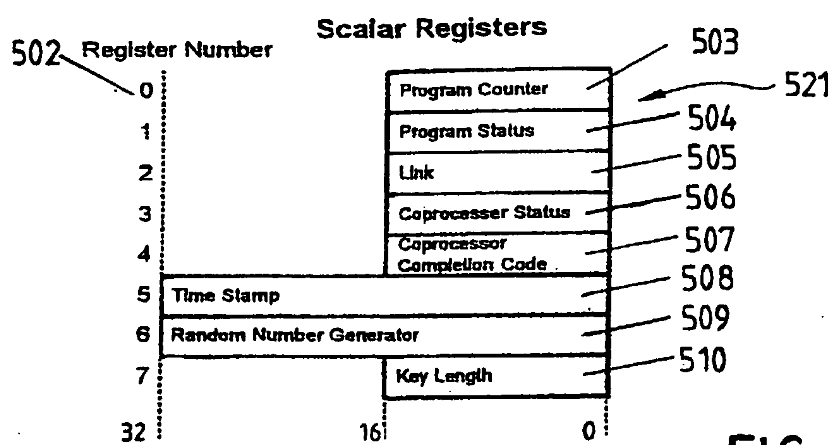


FIG. 5b

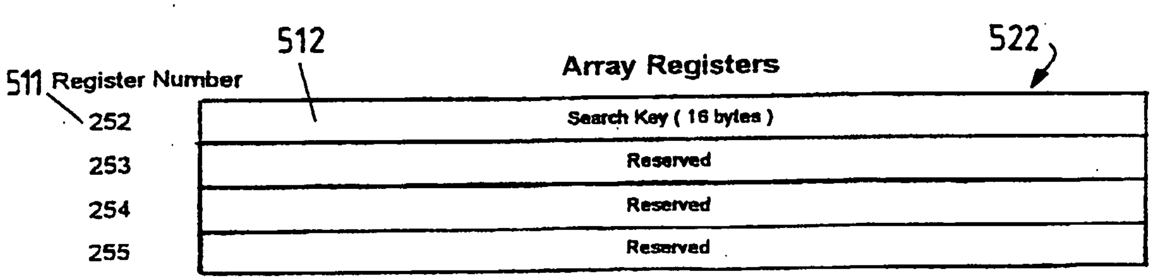


FIG. 5c

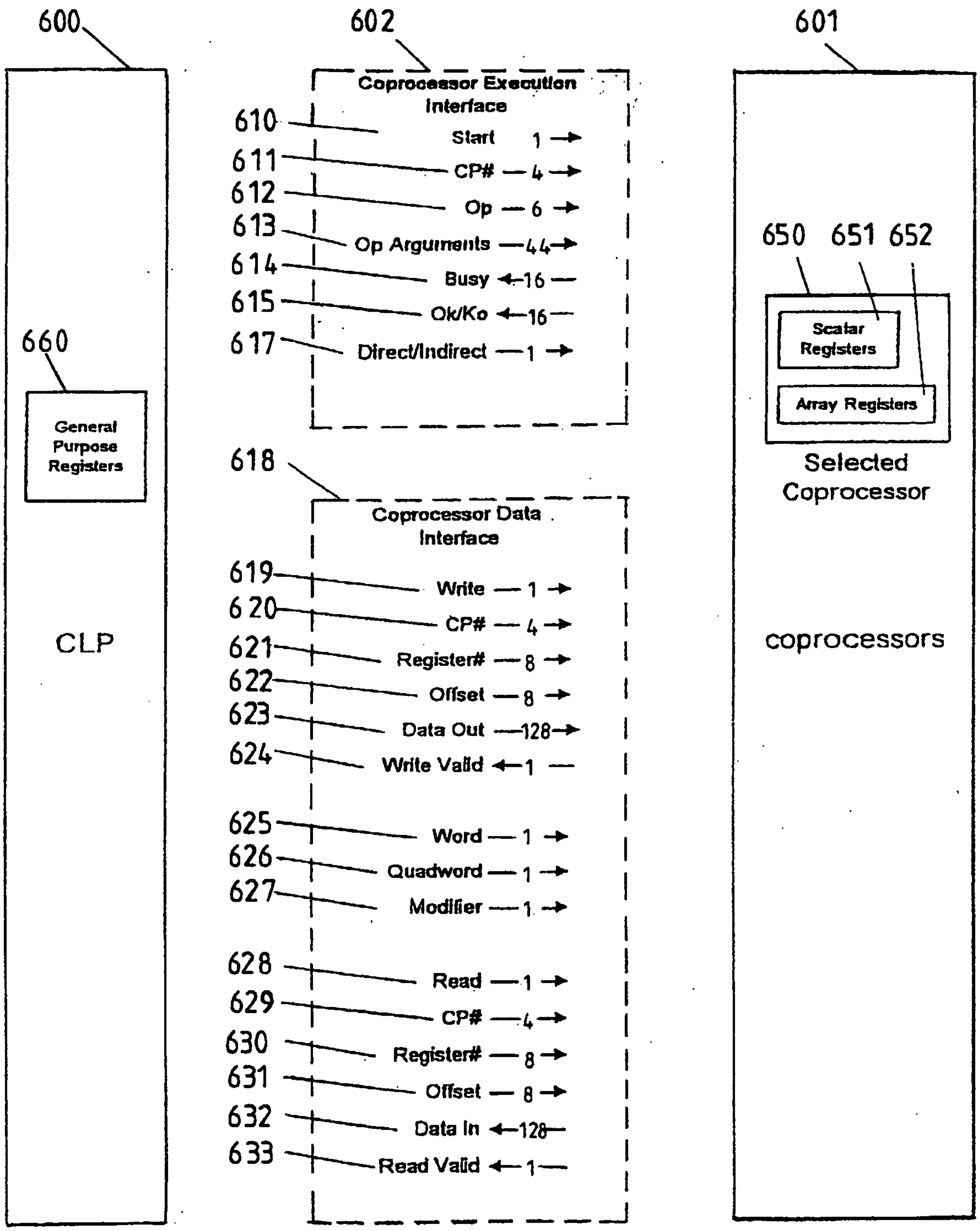

600
602
601
Coprocessor Execution Interface
610
Start 1
611
CP# 4
612
Op 6
613
Op Arguments 44
614
Busy 16
615
Ok/Ko 16
617
Direct/Indirect 1
650 651 652
Scalar Registers
Array Registers
Selected Coprocessor
660
General Purpose Registers
618
Coprocessor Data Interface
619
Write 1
620
CP# 4
621
Register# 8
622
Offset 8
623
Data Out 128
624
Write Valid 1
625
Word 1
626
Quadword 1
627
Modifier 1
628
Read 1
629
CP# 4
630
Register# 8
631
Offset 8
632
Data In 128
633
Read Valid 1
CLP
coprocessors

FIG. 6

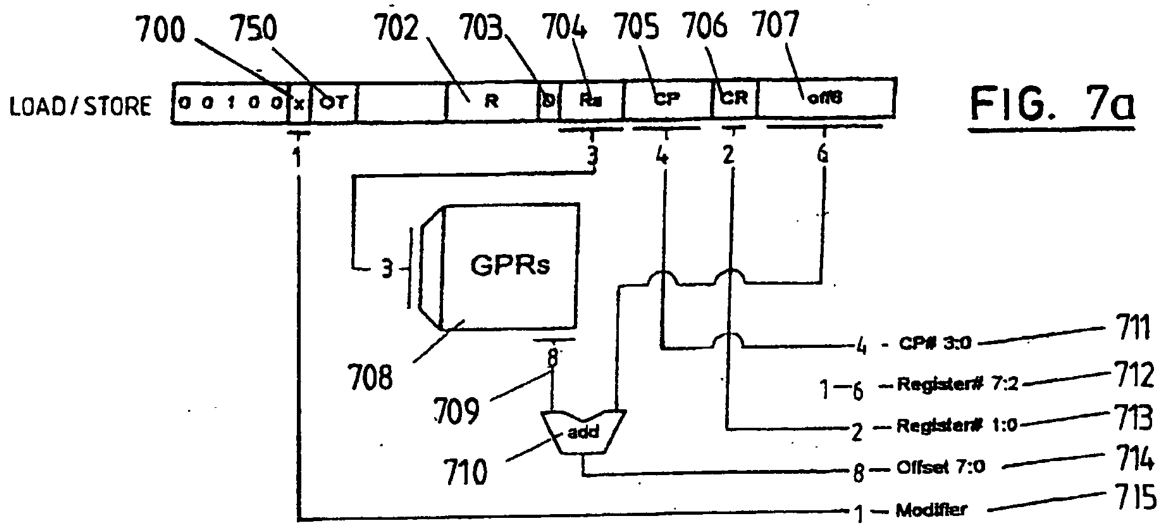


FIG. 7a

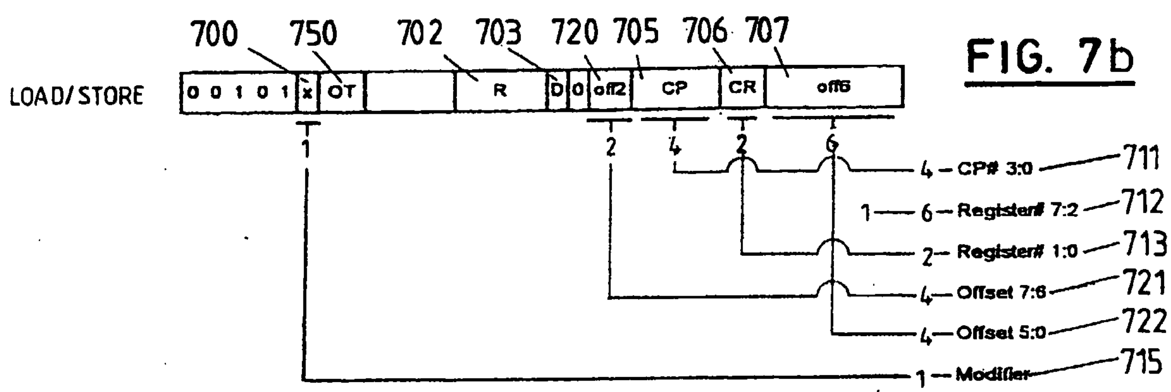


FIG. 7b

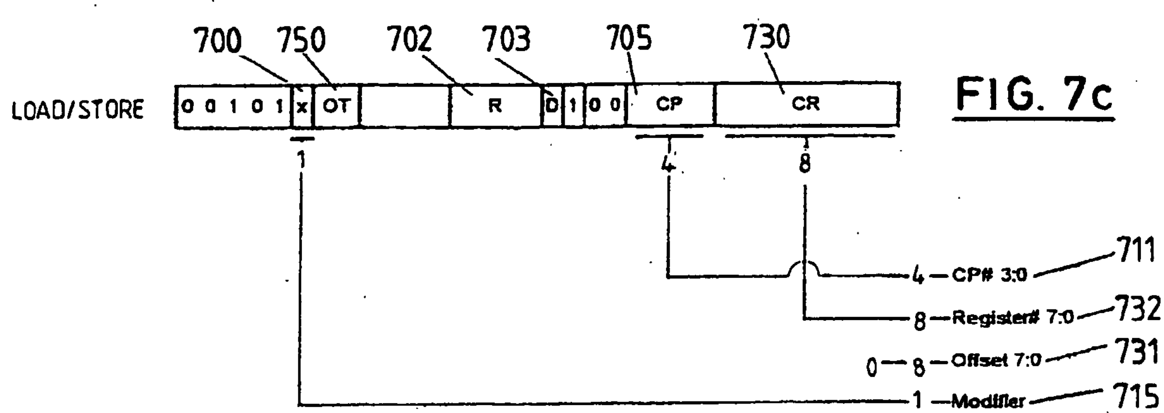


FIG. 7c

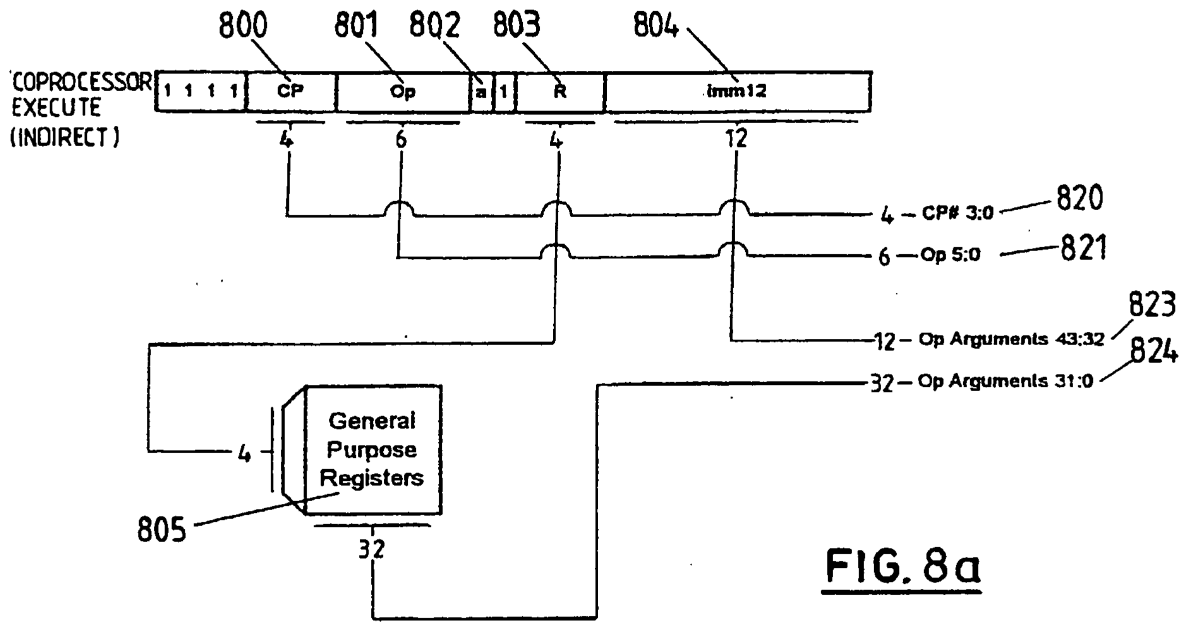

800
801
802
803
804
COPROCESSOR EXECUTE (INDIRECT)
1 1 1 1
CP
Op
a
1
R
imm12
4
6
4
12
4 – CP# 3:0
820
6 – Op 5:0
821
12 – Op Arguments 43:32
823
32 – Op Arguments 31:0
824
4
General Purpose Registers
805
32

FIG. 8a

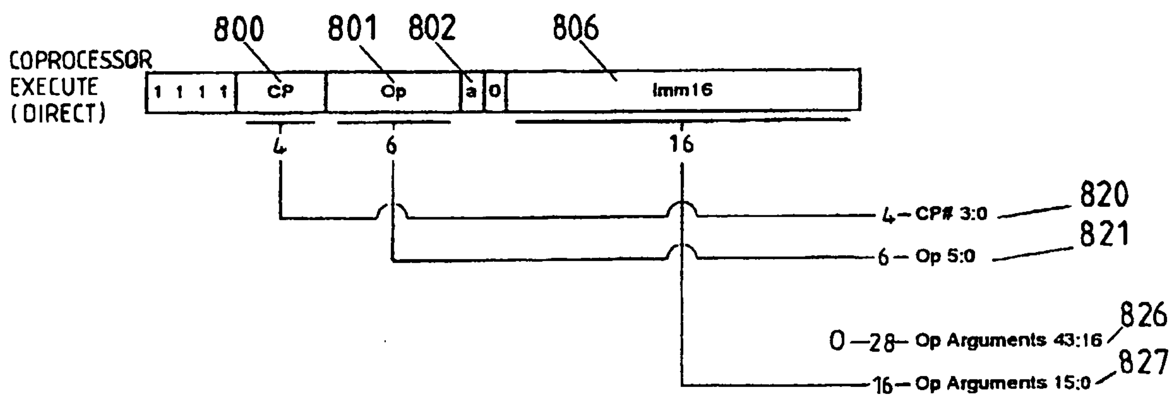

800
801
802
806
COPROCESSOR EXECUTE (DIRECT)
1 1 1 1
CP
Op
a
0
imm16
4
6
16
4 – CP# 3:0
820
6 – Op 5:0
821
0 – 28 – Op Arguments 43:16
826
16 – Op Arguments 15:0
827

FIG. 8b

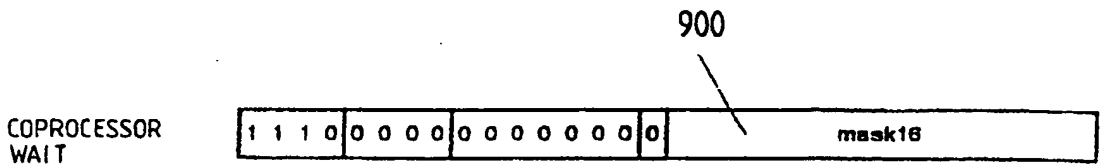

900
COPROCESSOR
WAIT
1 1 1 0
0 0 0 0
0 0 0 0 0 0 0
0
mask16

FIG. 9a

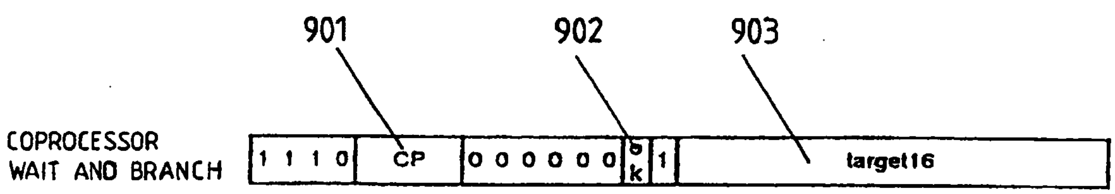

901
902
903
COPROCESSOR
WAIT AND BRANCH
1 1 1 0
CP
0 0 0 0 0 0
b
k
1
target16

FIG. 9b

2366426

COPROCESSOR DATA PROCESSING SYSTEM

The invention relates to a data processing system using coprocessor(s), and especially but not exclusive to network processors.

The use of a protocol processor unit (PPU) to provide for and to control the programmability of a network processor is well known. Likewise, the use of coprocessors with the PPU in the design of a computer system processing complex architecture is well established. Delays in processing events that require real time processing is a problem that directly affect system performance. By assigning a task to a specific coprocessor, rather than requiring the protocol processor unit to perform the task, a designer may increase the efficiency and performance of a computer system. Adding a coprocessor to a system under the prior art, requires the redesign of the hardware that provides the instructions required by the PPU to operate the coprocessor. However, a significant drawback to the efficient use of coprocessors is the need to redesign this hardware whenever a coprocessor is changed or added to the system.

According to the present invention there is provided a processing system comprising a main processing unit for executing a sequence of instructions in a stored program; at least one coprocessor unit responsive to said main processing unit, said at least one coprocessor unit being designed to perform specific task(s) under the control of said main processing unit; an interface between said processing unit and said at least one coprocessor unit and serving to enable one or more of the following functions for each such coprocessor unit: configuration of the coprocessor unit; initiation of specific task(s) to be completed by the coprocessor unit; access to status information relating to the coprocessor unit; and returning results relating to specific tasks completed by the coprocessor unit; said main processing unit and said at least one coprocessor unit each including one or more special purpose registers; said interface being capable of mapping said special purpose registers from said main processing unit and said at least one coprocessor unit into a common address map.

Typically, the main processing unit is a network processor, and each coprocessor unit is able to execute specific networking tasks. For example, one coprocessor unit computes CRC checksums. Another coprocessor unit moves blocks of data between local memory or array registers and a larger main memory. Another coprocessor unit searches a tree structure for data which corresponds to a specified key. One coprocessor unit assists in the enqueuing of packets once processing is complete. Still another coprocessor unit assists in accessing the contents of registers within said

processing system. Preferably the special purpose registers include scalar registers and array registers.

After initiating a task in a coprocessing unit, the main processing unit may either continue execution of instructions or it may stall the execution of further instructions until the completion of the task in the coprocessing unit. In the case where the main processing unit continues execution of instructions concurrent with task execution within the coprocessors, at some subsequent point in time, the execution of a WAIT instruction by the main processor unit will cause it to stall the execution of further instructions until the completion of task execution on one or more coprocessors. In one form, the WAIT instruction stalls execution on the main processing unit until task completion within one or more coprocessors at which time the main processing unit resumes instruction execution at the instruction following the WAIT instruction. In another form, the WAIT instruction stalls execution of the main processing unit until task completion within a specific coprocessor. When that task completes, the main processing unit examines a one bit return code from the coprocessor along with one bit from within the WAIT instruction to determine whether to resume instruction execution at the instruction following the WAIT instruction or branch execution to some other instruction specified by the programmer.

The main processing unit and the coprocessor unit each contains one or more special purpose scalar and array registers. These special purpose registers are mapped from the main processing unit and coprocessor units into a common address map.

An embodiment of the invention will now be described, by way of example, with reference to the accompanying drawings, in which:

Figure 1 shows the block diagram of a Protocol Processing Unit (PPU).

Figure 2 shows structure of a coprocessor's scalar registers.

Figure 3a shows the structure of a coprocessor's array registers.

Figure 3b illustrates addressing into an array register.

Figure 4 shows the complete instruction set for the core language processor (CLP).

Figure 5a shows the structure of the general purpose registers (GPRS) of the CLP.

Figure 5b shows the layout of the CLP's scalar registers, and Figure 5c shows the layout of the CLP's array registers.

Figure 6 describes the Coprocessor Execution Interface (CPEI) and the Coprocessor Data Interface (CPDI) which connects the CLP to its coprocessors.

Figures 7a, 7b, and 7c illustrate the Load / Store instruction formats.

Figures 8a and 8b illustrate the Coprocessor Execute instruction formats.

Figures 9a and 9b illustrate the Wait instruction formats.

In this embodiment, the invention will be described in terms of a Protocol Processor Unit (PPU) that provides and controls the programmability of a network processor. Referring to Figure 1, the PPU (100) comprises a Core Language Processor (CLP) (101) and 5 attached coprocessors (107, 108, 109, 110, 111). These coprocessors provide hardware acceleration for specific network processing tasks such as high speed pattern search, data manipulation, internal chip management functions, frame parsing, and data fetching.

Referring to Figure 1 the CLP (101) comprises an instruction fetch, decode, and execute unit (103) and a set of general purpose registers (104). The table in Figure 4 shows the CLP instruction formats which represent a set typical of a general purpose computer. They support:

Binary arithmetic operations add and subtract
Bit-wise Logical AND, OR, and NOT
Compare
Count leading zeros
Shift left/right Logical
Shift right arithmetic
Rotate Left and Right
Bit manipulation commands; Set, clear, test, and flip.
Loading a general purpose register with immediate data.
Branching.

Each instruction is 32 bits long. Instructions (400, 401, 402, 408, 409, 410, and 411) of Figure 4 relate to operations involving the coprocessors and are central to the embodiment. Again referring to Figure 1, the CLP fetches an instruction from instruction memory (102), and decodes it within its instruction decode unit (103) . With the exception of two instructions the CLP (101) completely executes the instruction within its execution unit (103). The two exceptions are the Coprocessor Execute (Direct) instruction (409) of Figure 4 and the Coprocessor Execute (Indirect) instruction (410) of Figure 4. These two instructions initiate command processing on one of the attached coprocessors. The coprocessors can execute commands concurrently with each other and concurrently with instruction processing within the CLP. Coprocessors provide two types of special purpose registers: scalar registers and array registers which are

described in more detail in Figures 2 and 3. Whenever a CLP instruction involves a coprocessor it specifies a 4-bit number called coprocessor identifier in the range 0 to 15 indicating which coprocessor is to be selected for the operation.

The embodiment of the invention contains 5 coprocessors. Referring to Figure 1 the following is a brief summary of each of these coprocessors:

A tree search engine (TSE) coprocessor (107), assigned coprocessor identifier 2. The TSE has commands for tree management and direct access to a tree search memory (112). It has search algorithms for performing searches for LPM (longest prefix match patterns requiring variable length matches), FM (fixed size patterns having a precise match) and SMT (software managed trees involving patterns defining either a range or a bit mask set) to obtain frame forwarding and alteration information. Details of a tree search architecture and operation useful in the present embodiment can be found in the following United States patent applications 09/543531, 09/544992 and 09/545100.

A data store coprocessor (109), assigned coprocessor identifier 1, for collecting, altering or introducing frame data into the network processor's frame data memory (113). Details are shown in U.S. patent application 09/384691.

The CAB coprocessor (111), assigned coprocessor identifier 3, provides the CLP with access to the Control Access Bus interface (CAB) (115). This bus provides access to the network processor's internal configuration and control registers. The architecture and operation of the CAB are shown in U.S. patent application 09/384691.

A conventional checksum coprocessor, assigned coprocessor identifier 5, to calculate and validate header checksums. Details are shown in U.S. patent application 09/384691.

An enqueue coprocessor (110), assigned coprocessor identifier 4, to enqueue frames to the network processor's various frame queues. Details are shown in U.S. patent application 09/384691.

The CLP (101) itself contains special purpose register unit (105) with scalar registers (116) and array registers (117) mapped within the address space assigned to coprocessor identifier 0. The CLP coprocessor (105) does not execute any commands.

Referring again to Figure 1, The CLP (101) is connected to its coprocessors (107, 108, 109, 110 and 111) via two interfaces: the Coprocessor Execution Interface (106) and the Coprocessor Data Interface (130). These interfaces are described in more detail in Figure 6.

As mentioned earlier the 4-bit coprocessor identifier uniquely identifies each coprocessor within the PPU (100) of Figure 1. Each coprocessor can support up to 256 special purpose registers. An eight bit register number in the range 0 to 255 uniquely identifies a special purpose register within a coprocessor. The combination of coprocessor number and register number uniquely identifies the register within the PPU. There are two types of special purpose registers: scalar registers and array registers.

Referring to Figure 2, the register numbers 0 (200) through 239 (202) are reserved for scalar registers. A scalar register (201) has a minimum length of 1 bit and a maximum length of 32 bits. Scalar register bits are numbered 0 through 31 starting with 0 at the rightmost or least significant bit and ending with 31 or the leftmost or most significant bit. Scalar registers of length less than 32 bits are right aligned and the remaining bits are considered unimplemented. When the CLP reads scalar registers of length less than 32 bits, the value of unimplemented bits is hardware dependent. Writing to unimplemented bits has no effect.

Referring to Figure 3a, the register numbers 240 through 255 are reserved for array registers. An array register has a minimum length of 2 bytes and a maximum length of 256 bytes. The CLP reads or writes an array register 2 bytes at a time (halfword), 4 bytes at a time (word) or 16 bytes at a time (quadword). Referring to Figure 3b, the CLP can read or write an array register beginning at any byte offset (304) including an odd byte offset. Addressing within an array register is modulo the length of the register. For instance, a quadword access to an n-byte long register beginning at offset n-1 affects the bytes at offsets n-1, 0, 1, and 2.

Figure 5 shows the layout of the general purpose registers (520) the scalar registers (521) and the array registers (522) within the CLP.

Referring to Figure 5a, the use of general-purpose registers is well-known in the art, and accordingly will be discussed in a general fashion. The general-purpose registers may be viewed by a programmer in two ways. A programmer may see a general purpose register as a thirty-two bit register, as is indicated by the thirty-two-bit labels w0 through w14 (500) which are represented with a 4 bit number from the set 0, 2, 4, … 14. In this sense the programmer sees eight 32-bit general purpose registers. A

programmer may also manipulate a general-purpose register as a sixteen-bit register, according to the sixteen-bit labels 501 r0 through r15 which are represented as a 4-bit number from the set 0, 1, 2, … 15. In this sense the programmer sees sixteen 16-bit registers.

Referring now to Figure 5b, the layout of the scalar registers (521) visible to a CLP programmer (103) are depicted. What are important in this embodiment is the coprocessor status register (506) and the coprocessor completion code register (507). The coprocessor status register (506) stores the information from the busy signal field (614) of Figure 6. This register indicates to a programmer whether a given coprocessor is available, or if it is busy. The coprocessor completion code register (507) stores information from the OK/K.O. field (615) of Figure 6. Therefore, if a programmer needs to know whether a given coprocessor is busy or is available, the programmer can get this information from the coprocessor status register (506). Similarly, the coprocessor completion code register (506) provides information to a programmer as to the completion of the coprocessor tasks.

The scalar register (521) provides for the following sixteen-bit program registers: a program counter register (503), a program status register (504), a link register (505), and a key length register (510). Two 32- bit registers are also provided: the time stamp register (508), and the random number generator register (509). A scalar register number (502) is also provided.

The general-purpose registers (520) may be viewed by a programmer in two ways. A programmer may see a general purpose register as a thirty-two bit register, as is indicated by the thirty-two-bit labels (500) shown in the Figure 5a (w0 through w14). A programmer may also manipulate a general-purpose register as a sixteen-bit register, according to the sixteen-bit labels (501) (r0 through r15).

The array registers (522) are revealed to a programmer through the array register numbers (511). Figure 5c depicts the layout of the array registers within the CLP.

Figure 6 depicts interface signals which connect the CLP (600) to its coprocessors (601). The Coprocessor Control Interface (106) of Figure 1 and the Coprocessor Data Interface (130) of Figure 1 are depicted in Figure 6 as (602) and (618) respectively. The number of individual wire connections is indicated by the numbering label appearing next to the arrow in each of the individual assignments. For the purposes of this discussion the selected coprocessor (650) represents the coprocessor whose coprocessor

identifier matches the coprocessor identifier appearing on either (611), (620), or (629) depending on the operation as described subsequently.

The execution interface (602) enables the CLP (600) to initiate command execution on any of the coprocessors (601). The coprocessor number 611 selects one of 16 coprocessors as the target for the command. When the CLP activates the start field (610) to logical 1, the selected coprocessor (650) as indicated by coprocessor number (611) begins executing the command specified by the 6-bit Op field (612). The op arguments (613) are 44 bits of data that are passed along with the command for the coprocessor (650) to process. The busy signal (614) is a sixteen-bit field, one bit for each coprocessor (601), and indicates whether a coprocessor is busy executing a command (bit = 1) or whether that coprocessor is not executing a command (bit = 0). These 16 bits are stored in scalar register (506) of Figure 5b where bit 0 of the register corresponds to coprocessor 0, bit 1 to coprocessor 1, etc. The OK/K.O. field (615) is a sixteen-bit field, one bit for each coprocessor (601). It is a one-bit return value code which is command specific. For example, it may be used to indicate to the CLP (600) whether a command given to a coprocessor (601) ended with a failure, or whether a command was successful. This information is stored within the CLP scalar register (507) in Figure 5b where bit 0 of the register corresponds to coprocessor 0, bit 1 to coprocessor 1, etc. The direct/indirect field (617) indicates to the selected coprocessor (650) which format of the Coprocessor Execute instruction is executing. If direct/indirect = 0, then direct format shown in Figure 9b is executing. else if direct/indirect = 1 then the indirect format shown in Figure 9a is executing.

The Coprocessor Data Interface (618) comprises 3 groups of signals. The write interface (619, 620, 621, 622, 623, 624) is involved in writing data to a scalar or array register within a coprocessor. The read interface (627, 628, 629, 630, 631, 632, 633) is involved in reading data from a scalar or array register within a coprocessor. The third group (625, 626, 627) is used during both reading and writing of a scalar register or array register. Duplicate functions on both read interface and write interface serve to support simultaneous read and write to move data from one register to another {e.g. interface signal (620) equivalent to signal (129)}.

The write interface uses the write field (619) to select a coprocessor (650) indicated by the coprocessor number (620). The write field (619) is forced to one whenever the CLP (600) wants to write data to the selected coprocessor. The coprocessor register identifier (621) indicates the register that the CLP (600) will write to within the selected coprocessor (650). The coprocessor register identifier (621) is an

eight-bit field and accordingly 256 registers are supported. A coprocessor register identifier the range 0 to 239 indicates a write to a scalar register. A coprocessor register identifier in the range 240 to 255 indicates a write to an array register. In the case of an array register write, the offset field (622) indicates the starting point for the data write operation in the array register. This field is eight-bits in size and therefore will support 256 addresses within an array. The data out field (623) carries the data that will be written to the coprocessor (650). It is 128 bits in size, and therefore up to 128 bits of information may be written in one time. The write valid field (624) indicates to the CLP (600) when the coprocessor (650) is finished receiving the data. This allows the CLP (600) to pause and hold the data valid while the coprocessor 650 takes the data.

The read interface is similar in structure to the write interface except that data is read from the coprocessor. The read field (628) corresponds to the write field (619), and is used by the CLP (600) to indicate when a read operation is to be performed on the selected coprocessor (650). The coprocessor number identifier field (629) determines which coprocessor (650) is selected. The register number field (630), offset field (631), and read valid field (633) correspond to (621) (622), and (624) in the write interface. The data-in field (632) carries the data from the coprocessor (650) to the CLP (600).

Read or write operations can have one of three lengths: halfword which indicates that 16 bits are to be transferred, word which indicates that 32 bits are to be transferred, and quadword which indicates that 128 bits are to be transferred. The read data 632 and the write data (623) are 128 bits in width. Data transfers of less than 128 bits are right aligned. Signals (625) and (626) indicate the data transfer size. 16-bit transfers are indicated by (625) and (626) both 0, 32-bits transfers are indicated by (625) and (626) being 1 and 0 respectively, and 128-bit transfers are indicated by (625) and (626) being 0 and 1, respectively.

The modifier field (627) is used during either a data read or data write operation. Each coprocessor interprets its meaning in its own fashion as defined by the coprocessor's hardware designer. It provides a way for the programmer to specify an additional bit of information to the hardware during either a read or write operation. The datestore coprocessor can skip the link field in the packet buffer in a linked list of packet buffers.

The following sections describe in greater detail the CLP instructions shown in Figure 4 that pertain to the interaction between the CLP 101 of Figure 1 and its coprocessors (107, 108, 109, 110, 111, and 105)

of Figure 1. These instructions are broken up into several categories: Load/Store, Coprocessor Execute, and Wait. Figures 7, 8, 9, and 10 show mapping between the bits in the various fields of the instructions and the interface signals shown in (602) and (618) of Figure 6. In this way it is demonstrated how the execution of specific CLP instructions (400, 401, 402, 408, 409, 410, and 411) of Figure 4 results in the activation of specific signals on the interfaces (602) and (618) of Figure 6.

Referring to Figure 4, instructions (400, 401, and 402) involve transferring data between the CLP's general purpose registers and a scalar or array register within a coprocessor. These instructions are shown in greater detail in Figure 7 and are referred to as the Load/Store instructions. Figure 7 shows the three different formats for the Load/Store instruction. Figure 7a and 7b are used to transfer data to or from an array register. Figure 7c shows the format used to transfer data to or from a scalar register. The general purpose register number field 702 specifies which general purpose register within the CLP (660) of Figure 6 will act as the source or destination of the data transfer. The data direction field D (703) determines the direction of this transfer as described in the following sections:

If field D (703) is equal to 0 then the data is copied from the selected coprocessor (650) of Figure 6 to the general purpose register (660) of Figure 6 specified by the general purpose register number field (702). In this case the signal (625, 626, 627, 628, 629, 630, 631, 632, and 633) of Figure 6 are used to perform the transfer. The signal (628) of Figure 6 is set to 1 indicating a read operation. The coprocessor identifier field (705) indicates the selected coprocessor via signal (629) of Figure 6. The data is transferred via signal (632) of Figure 6. The 2-bit operand type field (750) determines the width of the data to be copied as follows:

If field (750) is equal to 00 then general purpose register number field (702) specifies a 16-bit register as described in (500) of Figure 5a, signal (625) and (626) of Figure 6 are set to 0 and 0 respectively, causing 16-bits of data to be transferred from the selected coprocessor (650) of Figure 6 to the general purpose register (660) of Figure 6.

If field (750) is equal to 01 then general purpose register number field (702) is restricted to contain a number from the set 0, 2, 4, … 14 which specifies a 32-bit register as described in register (500) of Figure 5a. Signals (625) and (626) of Figure 6 are set to 1 and 0 respectively, causing 32-bits of data to be transferred from the

selected coprocessor (650) of Figure 6 to the general purpose register (660) of Figure 6.

The following describes the determination of the coprocessor register number (621) and (630) in Figure 6 which indicates which coprocessor register in the selected coprocessor (650) of Figure 6 participates in the above described data transfers.

Figure 7a and Figure 7b show the instruction formats for transferring data to or from an array register (652) of Figure 6 in the selected coprocessor (650) of Figure 6. In both instruction formats the coprocessor register number is determined by assigning the 2-bit field (706) to the low order 2 bits of the coprocessor register number (713). The high order 6 bits of the coprocessor register number (712) are set to 1. This restricts the coprocessor register number to be in the range 252-255. This is a limitation of the specific embodiment of the invention. Other embodiments could increase the size of the field (706) to 4-bits thereby allowing selection from the full set of array registers 240-255.

For data read operations (direction field (703) equal to 0), the coprocessor register number (712) and (713) indicate the selected coprocessor register via signal (630) of Figure 6. For data write operations (direction field (703) equal to 1) registers (712) and (713) indicate the selected coprocessor register via signal (621).

Continuing to refer to Figures 7a and 7c, the following describes the determination of the 8-bit array offset as described in (303) of Figure 3b which indicates which bytes from within the selected array register (652) of Figure 6 are to participate in the data transfer. Referring to figure 7a, the offset (707) to the low order 8 bits (709) of a 16-bit general purpose register selected from CPR (708). The selection is performed by using the 3-bit number specified by field (704) which selects from the set of 16-bit registers {r0, r1,---r7} described in 500 of Figure 5a. If field (704) equals 0 the r0 is selected if field (704) equals 1 then r1 is selected, etc. Referring to Figure 7b the full 8-it offset (721) and (722) is obtained from the instruction. The low order 6 bits (722) are obtained from (707) and the high order 2 bits (721) are obtained from (720). For data read operations (direction field 703 equal to 0), the offset (714) or (721) and (722) indicate the selected coprocessor array register offset via signal 631 of Figure 6. For data write operations (direction field (703) equal to 1), the offsets (714) or (721) and (722) indicate the selected coprocessor array register offset via signal 622 of Figure 6.

Figure 7c shows the instruction format for transferring data to or from a scalar register (651) of Figure 6 in the selected coprocessor (650) of Figure 6. Here a full 8-bit coprocessor register number (732) is obtained from instruction field (730). For data read operations (direction field (703) equal to 0), the coprocessor register number (730) indicates the selected coprocessor register via signal 630 of Figure 6. For data write operations (direction field (703) equal to 1) the coprocessor number (730) indicates the selected coprocessor register via signal (621) of Figure 6.

Instructions (411) and (410) of Figure 4 imitate command processing on a coprocessor by setting signal (610) of Figure 6 to a 1. Referring to Figure 8, the coprocessor identifier (820) is obtained from instruction field (800) and indicates the selected coprocessor (650) of Figure 6 via the start signal (611) of figure 6. The 6-bit coprocessor command is obtained from the instruction field (801) and indicates via signal (612) of Figure 6 to the selected coprocessor (650) of Figure 6 which command to begin executing. Upon activation of the start signal (610) of Figure 6 to a 1, the selected coprocessor 650 of Figure 6 activates to 1 its busy signal (614) of Figure 6 and keeps it at 1 until it completes execution of the command indicated by signal (612) of Figure 6 at which time it deactivates this signal to 0. The CLP (600) of Figure 6 continuously reads the 16 bits of signal (614) and places them into its Coprocessor Status Register (506) of Figure 5b. Upon completion of the command the selected coprocessor (650) of Figure 6 places this status in the appropriate bit of the Coprocessor Completion code register (507) of Figure 5b.

Referring once again to Figure 8, if the asynchronous execution field (802) of the instruction is 0 then the CLP (650) of Figure 6 indicates command completion by deactivating its busy signal (614). When this occurs, the CLP (600) of Figure 6 resumes fetching and execution of instructions. If the asynchronous execution field (802) of the instruction is 1 then the CLP (600) of Figure 6 continues fetching and execution of instructions regardless of the state of the busy signal (614) of Figure 6.

Upon initiation of command processing in the selected coprocessor (650) of Figure 6, the CLP (600) of Figure 6 supplies 44 bits of additional command specific information via signal (613) of Figure 6. This information is derived in one of two ways depending on the instruction format as depicted in Figures 8a and 8b.

The Coprocessor Execute indirect format of Figure 8a obtains the high order 12 bits (823) of command information from instruction field (804). The low order 32 bits of command information (824) are obtained from the

32-bit general purpose register selected from the register (805). The selected register is determined by the 4-bit instruction field (803) which is restricted to the values {0, 2, 4, …14}. In this way a 32-bit register from the set {w0, w2, w4,…….w14} is chosen as shown in register (500) of Figure 5a. The CLP (600) of Figure 6 sets signals (617) of Figure 6 to 1 indicating to the selected coprocessor (650) of Figure 6 that this is the indirect form of the instruction.

The Coprocessor Execute direct format of Figure 8b obtains the low order 16 bits (827) of the command information from instruction field (806). The high order 28 bits (826) of the command information are set to 0. The CLP (600) of Figure 6 sets signal 617 of Figure 6 to 0 indicating to the selected coprocessor (650) of Figure 6 that this is the direct form of the instruction.

The Coprocessor Execute direct format of Figure 8b obtains the low order 16 bits (827) of the command information from instruction field (806). The high order 28 bits (826) of the command information are set to 0. The CLP (600) of Figure 6 sets signal 617 of Figure 6 to 0 indicating to the selected coprocessor (650) of Figure 6 that his is the direct form of the instruction.

Instructions (408) and (409) of Figure 4 allow the CLP to wait for the completion command execution in one or more coprocessors.

Figure 9a depicts the instruction format for the Coprocessor Wait instruction (408) of Figure 4. The CLP (600) of Figure 6 performs the bit wise AND operation of the 16-bit mask obtained form instruction field (900) with the Coprocessor Status Register (506) of Figure 5b. If the result is not zero indicating that one or more coprocessors are still currently execution commands, the CLP (600) of Figure 6 stalls fetching and execution of instructions. However it continues to perform the above AND operation until which time the result is zero.

Figure 9b depicts the Coprocessor Wait and Branch format (409) of Figure 4. The coprocessor identifier field (901) indicates which specific bit in the Coprocessor Status Register (506) of Figure 5b is to be tested. For example if field (901) contains 1 then bit 1, of (506) of Figure 5b is tested. If (901) contains 15, then bit 15 of coprocessor status (506) in Figure 5b is tested. If the value of the tested bit is 1 indicating that the corresponding coprocessor has not yet completed command execution then the CLP (600) of Figure 6 stalls fetching and execution of instructions. However it continues to perform the above operation until the value of the tested bit is 0 indicating that the corresponding coprocessor has completed

command execution. At this time one of the two actions occur depending on the value of the ok field (902) of the instruction and the value of the bit in the Coprocessor Completion Code register (507) of Figure 5b as selected by the coprocessor identifier (901). The CLP (600) of Figure 6 either resumes fetching and execution at the next sequential instruction or it branches and resumes fetching and execution of instruction at the instruction address indicated by instruction filed (903) according to the following table:

Value of 902	Value of Selected Coprocessor Completion Code Bit = 0	Value of Selected Coprocessor Completion Code Bit = 1
0	branch	next instruction
1	next instruction	branch

The details of the instruction fetch, decode and execute unit within the CLP are known to persons of ordinary skill in the art.

CLAIMS

1. A processing system comprising:

a main processing unit for executing a sequence of instructions in a stored program;

at least one coprocessor unit responsive to said main processing unit, said at least one coprocessor unit being designed to perform specific task(s) under the control of said main processing unit;

an interface between said main processing unit and said at least one coprocessor unit and serving to enable one or more of the following functions for each such coprocessor unit:

configuration of the coprocessor unit;

initiation of specific task(s) to be completed by the coprocessor unit;

access to status information relating to the coprocessor unit; and

returning results relating to specific tasks completed by the coprocessor unit;

said main processing unit and said at least one coprocessor unit each including one or more special purpose registers;

said interface being capable of mapping said special purpose registers from said main processing unit and said at least one coprocessor unit into a common address map.

2. A processing system according to claim 1 wherein said main processing unit is a network processor, and the at least one coprocessor unit is able to execute specific networking tasks.

3. A processing system according to claim 1, 2 or 3 wherein said special purpose registers include scalar registers.

4. A processing system according to any one of claims 1 to 4, wherein said special purpose registers include array registers.

5. A processing system according to any preceding claim, wherein said main processing unit stalls execution of subsequent instructions upon initiating a task in a coprocessing unit, and said main processing unit resumes execution of subsequent instructions upon completion of said task by said coprocessing unit.

6. A processing system according to claim 5, wherein said main processing unit is programmed to execute a WAIT instruction after said coprocessor unit has completed said task in order to test the results of said task.

7. A processing system according to claim 6 wherein said WAIT instruction causes the program flow to branch if the results of said task are successful, and wherein said WAIT instruction causes the program flow to continue in line if the results of said task are not successful.

8. A processing system according to claim 6, wherein said WAIT instruction causes the program flow to branch if the results of said task are not successful; and wherein said WAIT instruction causes the program flow to continue in line if the results of said task are successful.

9. A processing system according to any one of claims 1 to 4, wherein said main processing unit is programmed to continue to execute subsequent instructions upon initiating a task in a coprocessing unit.

10. A processing system according to claim 9, wherein said main processing unit is programmed to execute a WAIT instruction after said coprocessor unit has started said task in order to suspend execution of subsequent instructions until said task is completed.

11. A processing system according to claim 9, wherein said main processing unit is programmed to execute a WAIT instruction after said coprocessor unit has started said task in order to test the results of said task when it is completed.

12. A processing system according to claim 11, wherein said WAIT instruction causes the program flow to branch upon completion of said task if the results of said task are successful and wherein said WAIT instruction causes the program flow to continue in line upon completion of said task if the results of said task are not successful.

13. A processing system according to claim 11, wherein said WAIT instruction causes the program flow to branch upon completion of said task if the results of said task are not successful and wherein said WAIT instruction causes the program flow to continue in line upon completion of said task if the results of said task are successful.

14. A processing system according to claim 2, wherein the said at least one coprocessor unit serves to compute CRC checksums.

15. A processing system according to claim 2, wherein the said at least one coprocessor unit serves to move blocks of data between local memory or array registers and a larger main memory.

16. A processing system according to claim 2, wherein the said at least one coprocessor unit serves to search a tree structure for data which corresponds to a specified key.

17. A processing system according to claim 2, wherein the said at least one coprocessor unit serves to assist in the enqueuing of packets once processing is complete.

18. A processing system according to claim 2, wherein the said at least one coprocessor unit serves to assist in accessing the contents of registers within said processing system.

INVESTOR IN PEOPLE

Application No:	GB 0108828.5	**Examiner:**	Richard Baines
Claims searched:	1 - 18	**Date of search:**	21 December 2001

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.S): G4A (AFGL, AMP, ANX)

Int Cl (Ed.7): G06F 9/38, 15/16, 15/17

Other: Online: EPODOC, WPI, JAPIO

Documents considered to be relevant:

Category	Identity of document and relevant passage		Relevant to claims
X	US 5,507,000	(BULL) - abstract, figure and column 5 lines 26 to 33.	1 & 9
X	US 4,626,634	(AT&T) - abstract, figure, column 5 lines 3 to 35 and column 7 line 64 to column 8 line 57.	1, 2 & 9

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
		E	Patent document published on or after, but with priority date earlier than, the filing date of this application.
&	Member of the same patent family		